

**8080/8085
FLOATING-POINT
ARITHMETIC LIBRARY
USER'S MANUAL**

Manual Order Number: 9800452-03



PREFACE

This manual describes Intel's 8080/8085 Floating-Point Arithmetic Library (FPAL) and its use. FPAL extends the capabilities of programs written for the 8080 and 8085 microcomputers. You can incorporate various floating-point operations into your 8080/8085 assembly-language or PL/M-80 program using simple procedure calls.

The manual includes programming examples in both languages, but assumes you already know how to use at least one of them. Programming information can be found in the following manuals.

8080/8085 Assembly Language:

8080/8085 Assembly Language Programming Manual, 9800301

ISIS-II 8080/8085 Assembler Operator's Manual, 9800292

PL/M-80:

PL/M-80 Programming Manual, 9800268

ISIS-II PL/M-80 Compiler Operator's Manual, 9800300



CONTENTS

CHAPTER 1 INTRODUCTION

	PAGE
What is FPAL?	1-1
Single-Precision Numbers	1-2
Integer Format	1-2
Floating-Point Format	1-2

CHAPTER 2 FLOATING-POINT RECORD PROCEDURES

FSET – Initialize Floating-Point Record	2-1
FRESET – Reset Error-Handling Procedure	2-3
FLOAD – Load FAC From Memory	2-3
FSTOR – Store Number into Memory From FAC	2-3
FSTAT – Access Status Information	2-4
FERROR – Access Error Information	2-4

CHAPTER 3 ARITHMETIC PROCEDURES

FADD – Floating-Point Addition	3-2
FSUB – Floating-Point Subtraction	3-2
FMUL – Floating-Point Multiplication	3-2
FDIV – Floating-Point Division	3-3
FSQRT – Floating-Point Square Root	3-3
FQFD2B – Decimal-to-Binary Conversion	3-4
FQFB2D – Binary-to-Decimal Conversion	3-5
FIXSD – Floating-Point to Integer Conversion	3-6
FLTDS – Integer to Floating-Point Conversion	3-6
FCMPR – Floating-Point Number Comparison	3-7
FZTST – Compare FAC to Zero	3-7
FNEG – Change Sign of FAC	3-8
FCLR – Clear FAC to Zero	3-8
FABS – Absolute Value	3-8

Sample Programs	3-8
8080 Assembly-Language Example	3-8
PL/M-80 Example	3-10

CHAPTER 4 ERROR HANDLING

Error-Handling Operation	4-1
FERHND – Default Error Handler	4-1
Error During Arithmetic Operation	4-1
Error During FQFD2B Operation	4-2
Error During FQFB2D Operation	4-2
Error During FIXSD Operation	4-2
Error During FCMPR Operation	4-2
Error During FZTST, FNEG, or FABS Operation ..	4-2
Other Calls to FERHND	4-2
Sample User Error Handlers	4-3
Assembly-Language Example	4-3
PL/M-80 Example	4-4

CHAPTER 5 INTERFACE TO FPAL

APPENDIX A FLOATING-POINT RECORD FORMAT

APPENDIX B DEFINITIONS

APPENDIX C SUMMARY OF FPAL PROCEDURES



ILLUSTRATIONS

FIGURE	TITLE	PAGE
2-1	Registers B, C Format for FSET	2-2
3-1	Control Block Format	3-4
A-1	Floating-Point Record Format	A-1
A-2	Floating-Point Number Format in Memory	A-3
A-3	Integer Format in Memory	A-3



TABLES

TABLE	TITLE	PAGE
C-1	FPAL Procedure Operation	C-1
C-2	FPAL Error-Handling Summary	C-2
C-3	FPAL Procedure Sizes	C-3

What is FPAL?

The Floating-Point Arithmetic Library (FPAL) contains basic floating-point subroutines and functions (referred to generically as 'procedures'). The operations provided are addition, subtraction, multiplication, division, value comparison, negation, clearing to zero, absolute value, square root, conversion between decimal and binary floating-point number representations, and conversion between floating-point and 32-bit signed integer formats. All operations are single precision (positive number range approximates 1.2×10^{-38} to 3.4×10^{38}). The single-precision format is described below and in Appendix B.

In addition to these operations, a number of procedures are provided to deal with the Floating-Point Record (FPR). This is a reserved, 18-byte work area used to collect status and error information, and as an accumulator for intermediate results. The procedures supporting the FPR perform FPR initialization, change error-recovery options, check the contents of FPR fields, and pass numbers between the FPR and memory.

The FPAL also includes a default error-handler subroutine. This subroutine is called when an invalid number is used in a floating-point operation or if overflow, underflow, or division by zero are not handled by an arithmetic subroutine. You may also write your own error handler, so long as it conforms to the formats described in this manual.

The FPAL can be used by assembly language or PL/M programs. The FPAL procedures reside in an ISIS-II library (FPAL.LIB) in object code form. They are self contained and can be used in component, OEM-board, or Intellec Microcomputer Development System environments.

In general, the following steps must be observed to use the floating-point library:

1. An area of memory must be reserved for the Floating-Point Record (FPR).
2. If your program uses interrupts or if you create your own stack, the appropriate amount of stack space must be allocated for use by FPAL, as described at the beginning of Chapter 3.
3. The names of the FPAL procedures you plan to use must be declared to be 'external' (using the EXTRN directive, in the ISIS-II 8080/8085 assembly language or the EXTERNAL attribute in PL/M-80).
4. FPAL procedure references must be imbedded in your source code where appropriate.
5. The FPAL procedure used by your program must be linked to your object file.

All FPAL procedures are reentrant and conform to PL/M-80 linkage conventions.

If you plan to reference FPAL procedures in your program, your program cannot use symbols that are reserved for FPAL. To avoid using these symbols inadvertently, do not use symbolic names beginning with a 'commercial at' sign (@) or names whose second character is 'Q' or '?'.

Single-Precision Numbers

FPAL procedures operate on single-precision binary numbers, either in a 32-bit integer format or in a 32-bit floating-point format.

Integer Format

The integer format recognized by FPAL is a positive or negative (two's complement) 32-bit binary number. The approximate range of this format is:

Decimal	Hexadecimal
+2.147 x 10 ⁹	7FFFFFFF
.	.
.	.
+0	00000000
-1	FFFFFFFF
.	.
.	.
-2.147 x 10 ⁹	80000000

Floating-Point Format

As an introduction to the single-precision floating-point format, consider the following representations of very small and very large decimal numbers. The decimal number base is used here to simplify the example.

Fixed-Point	Scientific Notation
6,373,000,000	6.373E+9 (6.373 x 10 ⁹)
0.00074	7.4E-4 (7.4 x 10 ⁻⁴)

The numbers in the two columns are equivalent. In the second column, the decimal point has been 'floated.' The exponent 'E' indicates the number of positions the decimal point was moved to the right or left to produce the abbreviated form shown. The numbers could have been written just as easily as '6373E+6' or '74E-5.'

The 32-bit, binary floating-point format recognized by FPAL consists of three fields:

sign	exponent	fraction
1 bit	8 bits	23 bits

The 'sign' field contains a *zero* if the number is non-negative and a *one* if the number is negative.

The 'exponent' field corresponds to the 'E' notation in the example above and indicates the number of bit positions the integer form of the number must be shifted to put it in the form '1.nnn'. Except in the case of floating-point zero, the value in the exponent field is offset by a *bias* of $2^7 - 1$ (or 127); i.e., the stored exponent is $(2^7 - 1)$ larger than the true exponent.

The 'fraction' field contains the 23 bits to the right of the binary point. An implicit one bit is assumed at the left of the binary point if the floating-point number is nonzero.

Example:

Integer	Floating-Point		
00000001 (hexadecimal)	0	011111110000...0000	
	sign	exp	fraction
	or, in hexadecimal:		
	3F800000		

The following lists make additional comparisons between decimal, binary integer, and binary floating-point representations. To save space, the internal binary representation is shown in hexadecimal form.

Decimal	Binary Integer	Binary Floating-Point
0	00000000 (hex)	00000000 (hex)
1	00000001	3F800000
-1	FFFFFFFF	BF800000
255	000000FF	437F0000
-255	FFFFFF01	C37F0000
*2.15 x 10 ⁹	7FFFFFF80 (note 1)	4EFFFFFF
*3.40 x 10 ⁻¹⁸		7F7FFFFFF (note 2)
*1.17 x 10 ⁻¹⁸		00800000 (note 3)
		40490FDB (π)
		7FFFFFFF (+infinity)
		FFFFFFFF (-infinity)

*approximately

NOTES

1. This is the largest integer that can be converted to single-precision floating-point retaining the exact value. Integer values within the interval [FF000000, 01000000] have exact single-precision floating-point representations. Integer values outside this interval may or may not be exact upon conversion, this conversion being implemented according to the "round to even" rule as described in Appendix B.
2. This is the largest number in the single-precision floating-point format.
3. This is the smallest positive number in the single-precision floating-point format.



CHAPTER 2 FLOATING-POINT RECORD PROCEDURES

If you plan to use FPAL procedures, you must allocate 18 contiguous bytes of memory for the Floating-Point Record (FPR). The FPR format is described in detail in Appendix A. In general, it is divided into four fields:

- Status field (1 byte).
- Error-Handler Address field (2 bytes). This is the address of the error recovery subroutine.
- Error field (2 bytes).
- Floating-Point Accumulator, or FAC. This consists of a fraction field (11 bytes) and an exponent field (2 bytes).

The remainder of this chapter describes the procedures used to initialize and access FPR fields. These procedures are:

FSET	A subroutine to initialize the FPR.
FRESET	A subroutine to reset the error-handling procedures and flags.
FLOAD	A subroutine to load a floating-point number from memory into the Floating-Point Accumulator (FAC) field of the FPR.
FSTOR	A subroutine to store a floating-point number from the FAC into memory.
FSTAT	A byte function that places the Status field of the FPR into the 8080 or 8085 accumulator.
FERROR	An address function that places the Error field of the FPR into 8080 or 8085 registers H and L.

The Floating-Point Record may be initialized and modified only by the procedures described here. The FSET initialization subroutine must be called before any other procedures are used; otherwise, the results are undefined.

The procedures described in this chapter save all 8080 or 8085 registers (except those registers receiving results from the operation called).

FSET—Initialize Floating-Point Record

This subroutine completes initialization of the FPR. To initialize the FPR, you must:

1. Push the address of the FPR onto the 8080 or 8085 stack;
2. Load register B with the error-handler indicator; load register C with the initial value for the Error Field;
3. Load registers D and E with the address of a user-defined error-handler subroutine, if necessary (see below);
4. Call FSET.

Before FSET is called, registers B and C should contain initial values as shown in figure 2-1. The shaded bits shown in this figure are reserved for FPAL use and should always be set to zero. Ones in these bit fields currently cause undefined results.

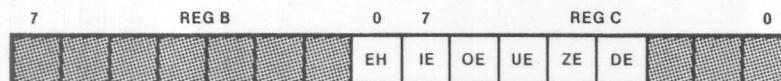


Figure 2-1. Registers B, C Format for FSET

The EH bit (register B) is interpreted as follows:

- EH = 0 The default error handler (FERHND) is to be used;
- EH = 1 Your own error handler is to be used and its address must be found in registers D and E.

If EH = 0, registers D and E are ignored. If EH = 1, FSET loads the contents of registers D and E into the Error-Handler Address field of the FPR.

NOTE

FSET always links an error handler named FERHND, whether you specify your own error-handling subroutine or not. If your own subroutine has the same name as the default subroutine, your error handler must appear before FPAL in the link list to ensure that your FERHND is linked instead of FPAL's.

LINK MYPROG.OBJ, FERHND.OBJ, FPAL.LIB...

FSET also clears the FAC and Status fields to zero and loads the contents of register C into the low-order byte of the Error field. See Appendix A for a detailed explanation of the register C bits.

In addition, FSET uses the STKLN assembler directive to allocate 40 bytes of stack for use by FPAL. This should be sufficient unless your program causes the FPAL routines to require multiple copies of stack storage. This could occur because of interrupt procedures which call FPAL, or because of user error handler recursion. Your program must add 40 extra bytes to STKLN for each level of FPAL invocation. (If you create your own stack, the number of bytes needed by FPAL is, likewise, $(40+40n)$, where n is the number of FPAL invocation levels.)

Examples:

The following 8080/8085 assembly-language sequence initializes the FPR and sets all bits in the Error field to zero. The example also assumes you are using the default error handler.

```
LXI    B,FPR      ;REGS B,C POINT TO FPR
PUSH   B          ;PUSH FPR ADDRESS ONTO STACK
LXI    B,0        ;USE DEFAULT ERROR HANDLER AND SET
                        ;REG C (ERROR FIELD) TO ZEROS
CALL   FSET       ;INITIALIZE FPR
```

In PL/M-80, the same operations can be done with the statement

```
CALL   FSET(.FPR,0,0);
```


FRESET—Reset Error-Handling Procedure

This subroutine is used to change the contents of the Error field or to specify that a different error handler be used. A common use of FRESET is to reset the five error flags in bits 3 - 7 of the Error field's low-order byte.

FRESET uses registers B and C in the same way as FSET (figure 2-1). If bit 0 of register B is one, registers D and E must contain the address of your error handler. The shaded bits in figure 2-1 should always be set to zero.

The FAC and Status Fields are not affected by FRESET.

Examples:

The following 8080/8085 assembly-language sequence clears the Error field mask bits to zero and specifies a user-defined error handler whose symbolic address is ERROR1. (Registers B and C are initialized separately to show clearly the specification of the error handler.)

```
LXI      B,FPR      ;REGS B,C POINT TO FPR
PUSH     B           ;PUSH FPR ADDRESS ONTO STACK
MVI      B,1        ;USE ERROR HANDLER ADDRESSED IN D,E
MVI      C,0        ;CLEAR ERROR FIELD TO ZEROS
LXI      D,ERROR1    ;POINTER TO ROUTINE ERROR1
CALL     FRESET      ;LOAD ERROR-RECOVERY INFORMATION
```

PL/M-80 statements to perform the same operation would be:

```
DECLARE   ERROR$FLAG LITERALLY '0000000100000000B';
CALL      FRESET (.FPR,ERROR$FLAG,ERROR1);
```

FLOAD—Load FAC from Memory

This subroutine loads a floating-point number from memory into the floating-point accumulator. FLOAD assumes that registers B and C contain the address of the FPR and that registers D and E address the low-order byte of the 32-bit number in memory.

Examples:

The following 8080/8085 assembly-language sequence loads a number, whose symbolic address is AUGEND, into the FAC.

```
LXI      B,FPR      ;REGS B,C POINT TO FPR
LXI      D,AUGEND    ;REGS D,E POINT TO 'AUGEND'
CALL     FLOAD       ;LOAD AND UNPACK 'AUGEND'
```

In PL/M-80, the same number is loaded by

```
CALL     FLOAD(.FPR,AUGEND);
```

FSTOR—Store Number into Memory from FAC

This subroutine stores the floating-point number in the FAC into memory. FSTOR assumes that registers B and C contain the address of the FPR and that registers D and E contain the address of the low-order byte of a 32-bit memory location.

Examples:

This 8080/8085 assembly-language example stores the contents of the FAC into the memory location addressed by RESULT.

```
LXI      B,FPR      ;REGS B,C POINT TO FPR
LXI      D,RESULT   ;REGS D,E POINT TO 'RESULT'
CALL     FSTOR      ;STORE FAC CONTENTS
```

The store is done in PL/M-80 by

```
CALL     FSTOR(.FPR,.RESULT);
```

FSTAT—Access Status Information

This function is called to access the contents of the FPR's Status field. FSTAT assumes the address of the FPR has been loaded into the B and C registers. When FSTAT is called, the contents of the Status field (one byte) are returned in the accumulator (register A).

Examples:

In 8080/8085 assembly language, the Status field is loaded by

```
LXI      B,FPR      ;REGS B,C POINT TO FPR
CALL     FSTAT      ;STATUS FIELD LOADED IN REG A
```

or, in PL/M-80,

```
DECLARE   STATFUN BYTE;
STATFUN = FSTAT (.FPR);
```

FERROR—Access Error Information

This function is called to access the contents of the FPR's Error field. It assumes the address of the FPR has been loaded into the B and C registers. FERROR returns the Error field contents (two bytes) to registers H and L.

Examples:

This 8080/8085 assembly-language example loads the contents of the Status and Error fields into the accumulator (register A) and into registers H and L, respectively.

```
LXI      B,FPR      ;REGS B,C POINT TO FPR
CALL     FSTAT      ;STATUS FIELD LOADED INTO REG A
CALL     FERROR     ;ERROR INFO TO REGS H,L
```

In PL/M-80, the corresponding operations would be:

```
DECLARE   STATFUN BYTE,
          ERRFUN ADDRESS;
STATFUN = FSTAT (.FPR);
ERRFUN = FERROR (.FPR);
```

This chapter describes the FPAL procedures for performing floating-point 'arithmetic.' These procedures are:

FADD	A subroutine to add floating-point numbers.
FSUB	A subroutine to do floating-point subtraction.
FMUL	A subroutine to multiply floating-point numbers.
FDIV	A subroutine to do floating-point division.
FSQRT	A subroutine to compute the square root of a floating-point number.
FQFD2B	A subroutine to convert a decimal floating-point number to binary.
FQFB2D	A subroutine to convert a binary floating-point number to decimal.
FIXSD	A subroutine to convert a floating-point number to an integer.
FLTDS	A subroutine to convert an integer to a floating-point number.
FCMPR	A byte function to compare floating-point numbers.
FZTST	A byte function to compare the FAC to zero.
FNEG	A subroutine to negate (change) the sign of the FAC.
FCLR	A subroutine to clear the FAC to zero.
FABS	A subroutine to set the FAC to its absolute value.

All of these subroutines assume that the B-C register pair contains the address of the FPR. If a second operand, stored in memory, is needed to perform an operation, the address of that operand's low-order byte is supplied in the D-E register pair. FCMPR and FZTST return their results to register A; FIXSD stores a fixed-point number into memory; FQFB2D stores a decimal floating-point number into memory; the other subroutines leave their results in the FAC.

The procedures described in this chapter, with the exception of FQFD2B and FQFB2D, save all 8080 or 8085 registers (except those registers receiving results from the arithmetic operation called).

The FPAL routines do not have their own stacks, but use the stack of the calling program. The FSET routine uses the STKLN assembler directive to allocate 40 bytes of stack for use by FPAL. This should be sufficient unless your program causes the FPAL routines to require multiple copies of stack storage. This could occur because of interrupt procedures which call FPAL, or because of user error handler recursion. Your program must add 40 extra bytes to STKLN for each level of FPAL invocation. If you create your own stack, the number of stack bytes needed by FPAL is, likewise, $(40+40n)$, where n is the number of FPAL invocation levels.

Appendix C summarizes all FPAL procedures and the error conditions they can return. Error handling is described in detail in Chapter 4.

NOTE

The FPR initialization subroutine (FSET) must be called before any of the arithmetic procedures can be used; otherwise, the results are undefined.

FADD—Floating-Point Addition

This subroutine adds a floating-point number in memory to the number in the Floating-Point Accumulator and leaves the sum in the FAC. FADD assumes that registers B and C contain the address of the FPR and that registers D and E address the low-order byte of the number in memory.

Examples:

8080/8085 assembly language:

```
LXI    B,FPR      ; REGS B,C POINT TO FPR
LXI    D,AUGEND    ; REGS D,E POINT TO 'AUGEND'
CALL   FLOAD      ; LOAD 'AUGEND' INTO FAC
LXI    D,ADDEND    ; REGS D,E POINT TO 'ADDEND'
CALL   FADD       ; ADD AUGEND AND ADDEND
LXI    D,SUM       ; REGS D,E POINT TO 'SUM'
CALL   FSTOR      ; STORE RESULT IN 'SUM'
```

PL/M-80:

```
CALL   FLOAD(FPR,,AUGEND);
CALL   FADD(FPR,,ADDEND);
CALL   FSTOR(FPR,,SUM);
```

FSUB—Floating-Point Subtraction

This subroutine subtracts a floating-point number in memory from the number in the Floating-Point Accumulator and leaves the result in the FAC. FSUB assumes that registers B and C contain the address of the FPR and that registers D and E address the low-order byte of the number in memory.

Examples:

8080/8085 assembly language:

```
LXI    B,FPR      ; REGS B,C POINT TO FPR
LXI    D,MINEND    ; REGS D,E POINT TO MINUEND
CALL   FLOAD      ; MINUEND LOADED INTO FAC
LXI    D,SBHEND    ; REGS D,E POINT TO SUBTRAHEND
CALL   FSUB       ; SUBTRACT SUBTRAHEND FROM MINUEND
LXI    D,RESULT    ; REGS D,E POINT TO 'RESULT'
CALL   FSTOR      ; STORE RESULT
```

PL/M-80:

```
CALL   FLOAD(FPR,,MINUEND);
CALL   FSUB(FPR,,SUBTRAHEND);
CALL   FSTOR(FPR,,RESULT);
```

FMUL—Floating-Point Multiplication

This subroutine multiplies the number in the Floating-Point Accumulator by a floating-point number in memory and leaves the product in the FAC. FMUL assumes that registers B and C contain the address of the FPR and that registers D and E address the low-order byte of the number in memory.

Examples:

8080/8085 assembly language:

```
LXI      B,FPR      ; REGS B,C POINT TO FPR
LXI      D,MPCAND    ; REGS D,E POINT TO MULTIPLICAND
CALL     FLOAD       ; MULTIPLICAND LOADED INTO FAC
LXI      D,MPLIER    ; REGS D,E POINT TO MULTIPLIER
CALL     FMUL        ; PERFORM MULTIPLICATION
LXI      D,PRODUCT   ; REGS D,E POINT TO 'PRODUCT'
CALL     FSTOR       ; STORE PRODUCT
```

PL/M-80:

```
CALL     FLOAD(.FPR,.MULTIPLICAND);
CALL     FMUL(.FPR,.MULTIPLIER);
CALL     FSTOR(.FPR,.PRODUCT);
```

FDIV—Floating-Point Division

This subroutine divides the number in the Floating-Point Accumulator by a floating-point number in memory and leaves the quotient in the FAC. FDIV assumes that registers B and C contain the address of the FPR and that registers D and E address the low-order byte of the number in memory.

Examples:

8080/8085 assembly language:

```
LXI      B,FPR      ; REGS B,C POINT TO FPR
LXI      D,DVDEND   ; REGS D,E POINT TO DIVIDEND
CALL     FLOAD       ; DIVIDEND LOADED INTO FAC
LXI      D,DIVSOR    ; REGS D,E POINT TO DIVISOR
CALL     FDIV        ; PERFORM DIVISION
LXI      D,QUOTNT    ; REGS D,E POINT TO 'QUOTNT'
CALL     FSTOR       ; STORE QUOTIENT
```

PL/M-80:

```
CALL     FLOAD(FPR ,DIVIDEND);
CALL     FDIV(.FPR,. DIVISOR);
CALL     FSTOR(.FPR,.QUOTIENT);
```

FSQRT—Floating-Point Square Root

This subroutine takes the square root of the number in the Floating-Point Accumulator and leaves the result in the FAC. FSQRT assumes that registers B and C contain the address of the FPR.

Examples:

8080/8085 assembly language:

```
LXI      B,FPR      ; REGS B,C POINT TO FPR
LXI      D,ARG       ; REGS D,E POINT TO ARGUMENT
CALL     FLOAD       ; ARGUMENT LOADED INTO FAC
CALL     FSQRT       ; COMPUTE SQUARE ROOT
LXI      D,RESULT    ; REGS D,E POINT TO 'RESULT'
CALL     FSTOR       ; STORE SQUARE ROOT RESULT
```


PL/M-80:

```

CALL    FLOAD(.FPR,.ARGUMENT);
CALL    FSQRT(.FPR);
CALL    FSTOR(.FPR,.RESULT);

```

FQFD2B—Decimal to Binary Conversion

This subroutine converts a decimal floating-point number in memory to a binary floating-point number and loads it into the FAC. FQFD2B assumes that registers B and C contain the address of the FPR and that registers D and E point to a 6-byte control block in memory. The control block, in turn, points to the decimal number to be converted. Before calling FQFD2B, you must define the control area and have the necessary information loaded into it.

The formats of the control block and decimal number are shown in figure 3-1. In this figure,

- SIGN** is the ASCII representation of '+' or '-'; FQFD2B assumes a '+' unless '-' is specified;
- SCALE** is a 16-bit, two's complement integer considered to be the exponent of ten;
- LENGTH** is an unsigned byte integer specifying the number of digits in the decimal number;
- ADDRESS** is a 16-bit address pointing to the first byte of the decimal number to be converted;
- D₁...D_n** are ASCII representations of decimal digits, and 'n' is the same as LENGTH.

The value of the number represented by this record is:

$$\text{SIGN}(D_1D_2\dots D_n) \cdot 10^{\text{SCALE}}$$

Zero is represented by setting all digits to zero or by setting LENGTH to zero.

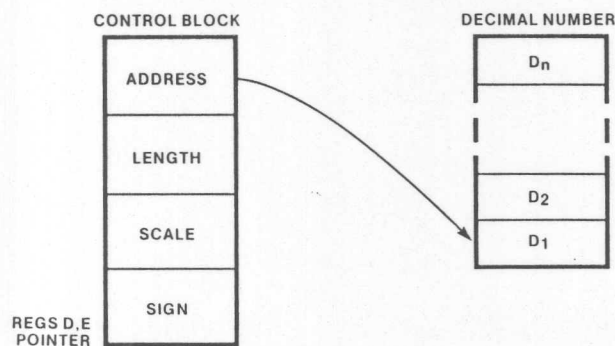


Figure 3-1. Control Block Format

Examples:

8080/8085 assembly language:

```

DSIGN:    DS 1           ; DEFINE CONTROL
DSCALE:   DS 2           ; BLOCK
DLNGTH:   DS 1
DADDR:    DS 2
;
; PROGRAM MUST SCAN DECIMAL NUMBER AND LOAD NECESSARY
; INFORMATION INTO CONTROL BLOCK
;
        LXI        B,FPR      ; REGS B,C POINT TO FPR
        LXI        D,DSIGN    ; REGS D,E POINT TO CONTROL BLOCK
        CALL       FQFD2B     ; CONVERSION DONE, RESULT STORED
                                ; IN FAC

```

PL/M-80:

```

DECLARE   CONTROL STRUCTURE(
          SIGN BYTE,
          SCALE ADDRESS,
          LENGTH BYTE,
          STRING$PTR ADDRESS),
          STRING (m) BYTE;
/*WHERE m IS GREATER THAN OR EQUAL TO CONTROL.LENGTH*/

/*PROGRAM MUST SCAN DECIMAL NUMBER AND LOAD NECESSARY*/
/*INFORMATION INTO CONTROL BLOCK*/

CALL FQFD2B(.FPR,.CONTROL);

```

FQFB2D—Binary to Decimal Conversion

This subroutine converts a binary floating-point number in the FAC to a decimal floating-point number and stores the result in memory. FQFB2D assumes that registers B and C contain the address of the FPR and that registers D and E point to a control block in memory. The control block has the format shown in figure 3-1 and points, in turn, to the memory location where the converted number is to be stored. At the time FQFB2D is called, you must also specify the contents of the LENGTH and ADDRESS fields of the control block.

The LENGTH field specification determines the precision of the result. The first digit (D_1) is nonzero unless the FAC contains zero.

Example:

8080/8085 assembly language:

```

; DEFINE STORAGE AS IN THE FQFD2B EXAMPLE ABOVE
;
DLNGTH    SET        10      ; LENGTH FIELD SPECIFIED
DADDR     SET        F0C8H   ; ADDRESS FIELD SPECIFIED
        LXI        B,FPR      ; REGS B,C POINT TO FPR
        LXI        D,DSIGN    ; REGS D,E POINT TO CONTROL BLOCK
        CALL       FQFB2D     ; CONVERSION DONE, RESULT STORED
                                ; IN MEMORY

```

PL/M-80:

```
/*DECLARE CONTROL BLOCK STRUCTURE AS IN THE*/
/*FQFD2B EXAMPLE ABOVE*/
```

```
/*ASSIGN POINTER TO SOME STRING ARRAY*/
CONTROL.STRING$PTR = .STRING;
/*ASSIGN VALUE FOR LENGTH OF STRING*/
CONTROL.SLENGTH = 10;
CALL FQFB2D(.FPR,.CONTROL);
```

FIXSD—FloatingPoint to Integer Conversion

This subroutine converts the floating-point (real) number in the FAC to a fixed-point (integer) number and stores the result in memory. This conversion is done with truncation (for example, 1.9 is converted to 1 and -1.9 is converted to -1). FIXSD assumes that registers B and C contain the address of the FPR and that registers D and E address the low-order byte of a 4-byte storage location. The resulting integer is stored in this location in two's complement format. See Appendix A, figure A-3.

Examples:

| *8080/8085 assembly language:*

```
LXI      B,FPR      ; REGS B,C POINT TO FPR
LXI      D,FLTNUM    ; REGS D,E POINT TO 'FLTNUM'
CALL     FLOAD       ; LOAD FLOATING-POINT NUMBER
LXI      D,FXNUM     ; ADDRESS FOR STORING RESULT
CALL     FIXSD       ; DO CONVERSION AND STORE RESULT
```

PL/M-80:

```
CALL     FLOAD(.FPR,.FP$NUMBER$ADDRESS);
CALL     FIXSD(.FPR,.INTEGER$ADDRESS);
```

FLTDS—Integer to Floating-Point Conversion

This subroutine converts a fixed-point number (32-bit signed integer) in memory to a floating-point number and loads the result into the Floating-Point Accumulator. Conversion is done using unbiased rounding (see Appendix B). FLTDS assumes that registers B and C point to the FPR and that registers D and E address the low-order byte of a 32-bit two's complement integer.

Examples:

| *8080/8085 assembly language:*

```
LXI      B,FPR      ; REGS B,C POINT TO FPR
LXI      D,FXNUM     ; REGS D,E POINT TO INTEGER
CALL     FLTDS       ; CONVERT INTEGER TO FLOATING-POINT
                        ; AND LOAD INTO FAC
```

PL/M-80:

```
CALL     FLTDS(.FPR,.INTEGER$ADDRESS);
```

FCMPR—Floating-Point Number Comparison

This function compares a number in the Floating-Point Accumulator to a floating-point number in memory. The resulting Status field settings are returned to the accumulator (register A). FCMPR assumes the B and C registers point to the FPR and that registers D and E address the low-order byte of the number in memory.

If the comparison is successful, one of the following bit patterns is set in the Status field and loaded into register A. ('U' means the bit is undefined and reserved for FPAL use.)

100UU000	FAC = number in memory
010UU000	FAC > number in memory
001UU000	FAC < number in memory

Examples:

8080/8085 assembly language:

```
LXI      B,FPR      ; REGS B,C POINT TO FPR
LXI      D,FACNUM    ; REGS D,E POINT TO 'FACNUM'
CALL     FLOAD       ; LOAD 'FACNUM' INTO FAC
LXI      D,MEMNUM    ; REGS D,E POINT TO 'MEMNUM'
CALL     FCMPR       ; NUMBERS COMPARED, STATUS TO REG A
```

PL/M-80:

```
CALL     FLOAD(.FPR,..FAC$NUMBER$ADDR);
STAT=FCMPR(.FPR,..MEMORY$NUMBER$ADDR);
```

FZTST—Compare FAC to Zero

This function compares the number in the Floating-Point Accumulator to zero and returns the Status field to the accumulator (register A). FZTST assumes that registers B and C address the FPR.

If the comparison is successful, one of the following bit patterns is set in the Status field and returned to register A. ('U' means the bit is undefined and reserved for FPAL use.)

100UU000	FAC = 0
010UU000	FAC > 0
001UU000	FAC < 0

Examples:

8080/8085 assembly language:

```
LXI      B,FPR      ; REGS B,C POINT TO FPR
LXI      D,TSTNUM    ; REGS D,E POINT TO TEST NUMBER
CALL     FLOAD       ; LOAD TEST NUMBER INTO FAC
CALL     FZTST       ; COMPARE NUMBER TO 0, STATUS TO REG A
```

PL/M-80:

```
CALL     FLOAD(.FPR,..TEST$NUMBER$ADDR);
STAT = FZTST(.FPR);
```


FNEG—Change Sign of FAC

This subroutine negates (complements) the sign bit of the FAC if the contents of the FAC are nonzero. A '1' bit is changed to '0' and vice-versa. If the number in the FAC is zero, no action is taken. FNEG assumes that registers B and C address the FPR.

Examples:

8080/8085 assembly language:

```
LXI      B,FPR      ; REGS B,C POINT TO FPR
LXI      D,NEGNUM    ; REGS D,E ADDRESS NUMBER WHOSE SIGN
                        ; IS TO BE NEGATED
CALL     FLOAD       ; LOAD 'NEGNUM'
CALL     FNEG        ; NEGATE SIGN OF 'NEGNUM'
```

PL/M-80:

```
CALL     FLOAD(.FPR,.NEGATE$NUMBER$ADDR);
CALL     FNEG(.FPR);
```

FCLR—Clear FAC to Zero

This subroutine clears the FAC by loading it with a floating-point zero (see Appendix B). FCLR assumes the B and C registers point to the FPR.

Examples:

8080/8085 assembly language:

```
LXI      B,FPR      ; REGS B,C POINT TO FPR
CALL     FCLR        ; THE FAC IS ZEROED
```

PL/M-80:

```
CALL     FCLR(.FPR);
```

FABS—Absolute Value

This subroutine sets the floating-point number in the FAC to its absolute value, that is, the sign bit is set to zero. FABS assumes the B and C registers address the FPR.

Examples:

8080/8085 assembly language:

```
LXI      B,FPR      ; REGS B,C POINT TO FPR
CALL     FABS        ; SIGN BIT SET TO ZERO
```

PL/M-80:

```
CALL     FABS(.FPR);
```

Sample Programs

8080 Assembly-Language Example

The following assembly-language example computes the weighted inner product

$$IP = (A1 * B1 + A2 * B2 + A3 * B3) / C1$$

A1, A2, A3, B1, B2, B3, and C1 represent addresses of floating-point numbers, FPR is the address of the Floating-Point Register and IP is the address where the result is to be stored.

First, we must reserve storage for the FPR and floating-point operands used in the equation. This is done with 'DS' assembler directive.

```
FPR:    DS      18
A1:     DS      4
B1:     DS      4
A2:     DS      4
B2:     DS      4
A3:     DS      4
B3:     DS      4
C1:     DS      4
IP:     DS      4
```

Next, we must declare the FPAL subroutines to be external using the 'EXTRN' directive.

```
EXTRN  FSET,FLOAD,FMUL,FADD,FDIV,FSTOR
```

The equation is then computed by the following sequence of loads and calls. Remember that FSET must be called before all other subroutines.

```
LXI      B,FPR      ; B,C POINTS AT THE FPR
PUSH     B
LXI      B,0         ; DEFAULT ERROR HANDLER TO BE USED
CALL     FSET        ; FPR IS INITIALIZED
LXI      B,FPR      ; POINTERS TO FPR AND A1 ARE LOADED
LXI      D,A1
CALL     FLOAD       ; A1 IS LOADED INTO THE FAC
LXI      D,B1        ; POINTER TO B1 IS LOADED
CALL     FMUL        ; A1 * B1 IS FORMED IN THE FAC
LXI      D,IP        ; POINTER TO IP IS LOADED
CALL     FSTOR       ; A1 * B1 STORED IN LOCATION ADDRESSED
                     ; BY IP

LXI      D,A2
CALL     FLOAD       ; A2 IS LOADED INTO THE FAC
LXI      D,B2
CALL     FMUL        ; A2 * B2 IS FORMED IN THE FAC
LXI      D,IP
CALL     FADD        ; A1 * B1 + A2 * B2 IS FORMED IN THE FAC
CALL     FSTOR       ; A1 * B1 + A2 * B2 IS STORED IN IP

LXI      D,A3
CALL     FLOAD       ; A3 IS LOADED INTO THE FAC
LXI      D,B3
CALL     FMUL        ; A3 * B3 IS FORMED IN THE FAC
LXI      D,IP
CALL     FADD        ; A1 * B1 + A2 * B2 + A3 * B3 IS FORMED IN
                     ; THE FAC

LXI      D,C1
CALL     FDIV        ; (A1 * B1 + A2 * B2 + A3 * B3)/C1 IS FORMED
                     ; IN THE FAC

LXI      D,IP
CALL     FSTOR       ; (A1 * B1 + A2 * B2 + A3 * B3)/C1 IS STORED
                     ; IN IP
```

This example assumes the default error handler (FERHND) is to be used. At the end of the computation, you can check to see whether any errors occurred by executing the following code sequence:

```
CALL      FERROR      ; THE CUMULATIVE ERROR INDICATORS
                        ; ARE RETURNED IN H,L
MOV       A,L
ANI       1111000B    ; MASK OFF THE OPTION BITS
JNZ       HELP        ; AT LEAST ONE ERROR OCCURRED
```

PL/M-80 Example

The following PL/M-80 example computes the same weighted inner product as the assembly-language example:

$$IP = (A1 * B1 + A2 * B2 + A3 * B3) / C1$$

A1, A2, A3, B1, B2, B3, and C1 represent addresses of floating-point numbers, FPR is the address of the Floating-Point Register and IP is the address where the result is to be stored.

We must first declare the FPAL subroutines used to be external procedures and reserve the FPR memory area as an array. Declaring the operators to be arrays too ensures that they will occupy contiguous locations in memory, thus allowing use of the dot operator in calling the subroutines. For the sake of illustration, the FSTAT function is also included in this example.

```
/*DEFINE EXTERNAL PROCEDURES*/

FSET:      PROCEDURE (FA,OP1,OP2) EXTERNAL;
           . DECLARE(FA,OP1,OP2) ADDRESS;
END FSET;

FADD:      PROCEDURE(FA,OA) EXTERNAL;
           . DECLARE(FA,OA) ADDRESS;
END FADD;

FDIV:      PROCEDURE(FA,OA) EXTERNAL;
           . DECLARE(FA,OA) ADDRESS;
END FDIV;

FMUL:      PROCEDURE(FA,OA) EXTERNAL;
           . DECLARE(FA,OA) ADDRESS;
END FMUL;

FLOAD:     PROCEDURE(FA,OA) EXTERNAL;
           . DECLARE(FA,OA) ADDRESS;
END FLOAD;

FSTOR:     PROCEDURE(FA,OA) EXTERNAL;
           . DECLARE(FA,OA) ADDRESS;
END FSTOR;

FSTAT:     PROCEDURE(FA) BYTE EXTERNAL;
           . DECLARE(FA) ADDRESS;
END FSTAT;

/*DECLARE BYTE ARRAYS*/
```

```
DECLARE   FPR(18)      BYTE,  
          A1(4)        BYTE,  
          A2(4)        BYTE,  
          A3(4)        BYTE,  
          B1(4)        BYTE,  
          B2(4)        BYTE,  
          B3(4)        BYTE,  
          C1(4)        BYTE,  
          IP(4)        BYTE,  
          STATUS      BYTE;
```

```
/*IP COMPUTED BY FOLLOWING CALLS*/  
/*FSET MUST BE CALLED FIRST*/
```

```
CALL      FSET(.FPR,0,0);/*USE FERHND*/  
CALL      FLOAD(.FPR,.A1);  
CALL      FMUL(.FPR,.B1);  
CALL      FSTOR(.FPR,.IP);  
CALL      FLOAD(.FPR,.A2);  
CALL      FMUL(.FPR,.B2);  
CALL      FADD(.FPR,.IP);  
CALL      FSTOR(.FPR,.IP);  
CALL      FLOAD(.FPR,.A3);  
CALL      FMUL(.FPR,.B3);  
CALL      FADD(.FPR,.IP);  
CALL      FDIV(.FPR,.C1);  
CALL      FSTOR(.FPR,.IP);
```

```
/*RETURN STATUS FIELD*/
```

```
STATUS=FSTAT(.FPR);
```

Error-Handling Operation

When an error occurs during an FPAL operation, the following steps are taken:

1. The address of the FPR is pushed onto the 8080 or 8085 stack.
2. A code is placed in the B-C register pair indicating which procedure was executing when the error was detected.
3. The error code bits in the FPR's Status field are set to indicate the type of error detected.
4. The appropriate cumulative error bit in the FPR's Error field is set.
5. The error-handler subroutine is called.

The bit settings mentioned in steps 2, 3, and 4 are listed in Appendix C, Table C-2.

If the executing procedure required a second operand, that operand's address is in the D-E register pair. Otherwise, the D-E register pair is ignored.

FERHND—Default Error Handler

This subroutine is the error handler supplied as part of the floating-point library. You may also write your own error handler and load its address using the FSET or FRESET subroutines (Chapter 2).

The operations performed by FERHND depend on the parameters passed to it by the procedure detecting the error, and on the error code found in the Status field. These operations are described in the following paragraphs and summarized in Appendix C, Table C-2. Note that FERHND may find it necessary to update the error code (the three least significant bits in the Status field) to reflect the current true status of the Floating-Point Accumulator.

Error During Arithmetic Operation

If FERHND was called during one of the four basic arithmetic operations (FADD, FSUB, FMUL, FDIV) one of the following situations occurs:

- If underflow is indicated, the FAC is set to zero and the Status field is set to 'UUUUU000,' where 'U' means the bit setting is undefined.
- If overflow is indicated, the FAC is set to the largest or smallest representable number (if the correct result was positive or negative, respectively). The Status field is set to 'UUUUU000.'
- If division by zero was attempted, the FAC is set to an invalid number representing an 'indefinite' result. The 's' bit is zero, all exponent bits are one, and all fraction bits are zero. The Status field is set to 'UUUUU101.'
- If an invalid operand was encountered, no operation is performed and FERHND returns to the calling subroutine.
- If none of these conditions holds, FERHND simply returns to the calling subroutine.

Error During FQFD2B Operation

The FQFD2B procedure does not check for valid ASCII representations in the input operand. If invalid data is used, no error conditions are reported, but the result is undefined.

Overflow or underflow may occur during the conversion. In this case the error is regarded as an arithmetic error and the error is handled as described in the preceding section.

Error During FQFB2D Operation

As in the case of FQFD2B, overflow or underflow errors may result from an arithmetic operation within the conversion procedure. These errors are handled by the arithmetic procedure involved.

If the FAC contains an invalid quantity when FQFB2D is called, this procedure stores an asterisk (*) in the SIGN position of the decimal representation (see figure 3-1) and in digit positions D_2 through D_n . One of the following codes is stored in the first digit position (D_1):

- + if the FAC contains +INF
- if the FAC contains -INF
- ? if the FAC contains IND
- 0 if the FAC contains -0
- * if the FAC contains any other invalid quantity.

'INF' and 'IND' are defined in Appendix B.

Error During FIXSD Operation

If FERHND is called by FIXSD, one of the following occurs:

- If overflow is indicated (number in FAC too large to be converted to a 32-bit integer), the result is set to the largest positive or negative integer (if the number in the FAC is positive or negative, respectively). The FPR remains unchanged except that the Status field is set to 'UUUUU000.'
- If the number in the FAC is invalid, FERHND simply returns. The integer stored by FIXSD is undefined.

Error During FCMPR Operation

If FERHND is called by FCMPR, at least one of the operands must be invalid. If the operands are identical invalid bit patterns, the Status field is set to '100UU101.' Otherwise, the Status field is '000UU101.'

Error During FZTST, FNEG, or FABS Operation

If the calling procedure is FZTST, FNEG, or FABS, no operation is performed and the error handler simply returns.

Other Calls to FERHND

If FERHND is called from somewhere other than the floating-point procedures listed above, the result is undefined.

Sample User Error Handlers

If you write your own handler and use FPAL arithmetic subroutines, be aware that your error handler may be called recursively, and must therefore be reentrant. Since FPAL does not have its own stack, you must allocate 40 bytes of your own program stack for each level of recursion foreseen.

If you are writing your error handler in PL/M, it must be written and called with three parameters (although the last parameter may actually be a dummy).

Assembly-Language Example

The following is an example of a reentrant error-recovery routine (ERREC). If the calling program is FADD, FSUB, FMUL, or FDIV, and if the error condition is underflow, the result is set to zero. Otherwise, the error-recovery routine returns.

The address of the low-order byte of the Floating-Point Record is assumed to be on the stack and the B-C register pair is assumed to contain the code indicating which procedure called ERREC. If the procedure required two operands, the second operand's address is assumed to be in the D-E register pair.

```

NAME          ERREC
CSEG
PUBLIC        ERREC
EXTRN         FCLR, FSTAT

;
; SAVE THE REGISTER CONTENTS
;
PUSH          PSW
PUSH          B
PUSH          H

;
; MOVE THE ERROR CODE TO 'A'. LOAD THE POINTER TO THE FPR INTO
; B,C AND MOVE THE RETURN ADDRESS TO WHERE THE POINTER WAS
;
MOV           A,C
PUSH          D
LXI           H,8
DAD           SP
MOV           E,M
INX           H
MOV           D,M
INX           H
MOV           C,M
INX           H
MOV           B,M
MOV           M,D
DCX           H
MOV           M,E
POP           D

;
; THE CODE SETTINGS IN 'A' DESIGNATE WHICH PROCEDURE CALLED
; THE ERROR RECOVERY ROUTINE
;
; A = 1 : FADD
; A = 2 : FSUB
; A = 3 : FMUL
; A = 4 : FDIV
; A = 5 : FIXSD
; A = 6 : FCMPR

```

```

; A = 7 : FZTST
; A = 8 : FNEG
; A = 9 : FABS
; A = 10 : FSQRT
;
; IF A = 1,2,3,4 AND IF THE ERROR CONDITION IS UNDERFLOW,
; SET THE RESULT TO ZERO. OTHERWISE, SIMPLY RETURN.
;
      CPI      5
      JNC      DONE
      CALL     FSTAT
      ANI      00000111B
      CPI      4
      JNZ      DONE
      CALL     FCLR
;
; RESTORE REGISTERS AND STACK
;
DONE:  POP      H
      POP      B
      POP      PSW
      INX      SP
      INX      SP
      RET
      END

```

PL/M-80 Example

The following code tells the FPAL that a user routine (USER\$ERROR) is to be called when an error is detected and loads the address of the error routine into the FPR. If the calling procedure required two operands, the second operand's address is passed as the third parameter of USER\$ERROR.

```

DECLARE ERROR$FLAG LITERALLY '0000000100000000B';
CALL FSET(FPR,ERROR$FLAG,USER$ERROR);

```

The remainder of this example is code needed to print a message indicating which procedure was running when the error occurred.

```

WRITE      PROCEDURE (AFT, BUFFER, COUNT, STATUS) EXTERNAL;
      DECLARE (AFT, BUFFER, COUNT, STATUS) ADDRESS;
END WRITE;

```

```

USER$ERROR: PROCEDURE (FPR, ERROR, ADDR);
      DECLARE (FPR, ERROR, ADDR, STATUS) ADDRESS;
      ;
      ; DO CASE ERROR;
      ;
      CALL WRITE (0,('FADD ERROR '),11,STATUS);
      CALL WRITE (0,('FSUB ERROR '),11,STATUS);
      CALL WRITE (0,('FMUL ERROR '),11,STATUS);
      CALL WRITE (0,('FDIV ERROR '),11,STATUS);
      CALL WRITE (0,('FIXSD ERROR '),12,STATUS);
      CALL WRITE (0,('FCMPR ERROR '),12,STATUS);
      CALL WRITE (0,('FZTST ERROR '),12,STATUS);
      CALL WRITE (0,('FNEG ERROR '),11,STATUS);
      CALL WRITE (0,('FABS ERROR '),11,STATUS);
      CALL WRITE (0,('FSQRT ERROR '),12,STATUS);
      END;
END USER$ERROR;

```

The FPAL procedures reside in object module form in the library `FPAL.LIB` on the ISIS-II system diskette. You need only declare the names of the FPAL procedures you use to be 'external' and call them when they are needed. When you have completed program development, you must link the necessary floating-point procedures to your object module.

FPAL procedure names are declared to be external using the `EXTRN` directive in assembly language or the `EXTERNAL` attribute in PL/M. The simplest way to do this is to create a file containing external declarations for the FPAL procedures you will be using, then incorporate this file into your source program using the `INCLUDE` control in the 8080/8085 assembler or PL/M-80 compiler. For example, you might imbed the `INCLUDE` control in your source code as follows:

```
$INCLUDE (:F1:FPEXTN.SRC)
```

Since the FPAL procedures reside in an ISIS-II library, they can be linked quite easily by linking the entire library. The linker then scans your program and links only those procedures you need (those that satisfy external references). Linking is done at the ISIS-II command level following successful assembly/compilation to produce a relocatable 8080 or 8085 object module. The PL/M-80 library (`PLM80.LIB`) must be linked.

Example:

```
-LINK :F1:MYPROG.OBJ,FPAL.LIB,PLM80.LIB TO :F1:MYPROG.LNK
```

You can also specify individually the FPAL procedures you want linked from `FPAL.LIB`. If you choose to let the linker satisfy external references, you should be sure you do not have external declarations for procedures you don't use. For example, you would not want to create an 'include' file containing external declarations for *all* FPAL procedures unless you plan to specify individual 'modules' at the time you link `FPAL.LIB`, or intend to use all of them.

If your set of program modules includes PL/M-80 or 8080/8085 assembly language modules that call FPAL routines and also includes other modules written in FORTRAN-80 (none of which should call FPAL routines), you should link your program modules together (after translating) using the three-step sequence given below, rather than as illustrated in the example above. This is necessary to avoid incorrect references to certain routines in the FORTRAN-80 run-time libraries that have the same public names as routines in `FPAL.LIB`.

1. Link together your FORTRAN relocatable object program(s) and the support libraries they need, including the FORTRAN-80 run-time libraries required for your environment (refer to the *ISIS-II FORTRAN-80 Compiler Operator's Manual*), but omitting `PLM80.LIB`. The `LINK` output from this step will include a list of unresolved external names of routines in `PLM80.LIB`, and also the unresolved external names of any PL/M or assembly language routines called from your FORTRAN program(s). These can be ignored.
2. Link together your PL/M-80 or 8080/8085 assembly language relocatable object program(s) and the support libraries they need, but again omitting `PLM80.LIB`. The `LINK` output will again include a list of unresolved external names of routines in `PLM80.LIB`, and also the unresolved external names of any FORTRAN routines called from your PL/M or assembly language program(s). These, again, can be ignored.

3. Link all modules together in this order if your main program is in FORTRAN:
(LINK output from step 1), (LINK output from step 2), PLM80.LIB
Or, link all modules in this order if your main program is in PL/M or assembly language:
(LINK output from step 2), (LINK output from step 1), PLM80.LIB
The LINK output from this step will include messages that certain symbols beginning with the letters "FQ" are multiply defined. These messages can be ignored; the pre-linking steps (1 and 2) have ensured that the correct routines are linked to the calling modules that need them.

Example:

Your main program is written in FORTRAN-80, uses FORTRAN intrinsic functions (so that FPEF.LIB is needed), and is stored in relocatable object form in FTNMN.OBJ. The main program calls two PL/M-80 subroutines that call FPAL routines and also make ISIS-II system calls (so that SYSTEM.LIB is needed). The object code for these subroutines is in PLMSB1.OBJ and PLMSB2.OBJ. You want to run your programs under ISIS-II and use the software run-time package for FORTRAN arithmetic. If all the Intel libraries are on drive 0 and all your own program modules are on drive 1, you can use the following sequence of LINK commands:

1. —LINK :F1:FTNMN.OBJ,F80RUN.LIB,F80ISS.LIB, &
**FPEF.LIB,FPSOFT.LIB TO :F1:FTNMN.LNK
2. —LINK :F1:PLMSB1.OBJ,:F1:PLMSB2.OBJ,FPAL.LIB, &
**SYSTEM.LIB TO :F1:PLMSB.LNK
3. —LINK :F1:FTNMN.LNK,:F1:PLMSB.LNK,PLM80.LIB &
**TO :F1:MYPROG.LNK

The Floating-Point Record is allocated as shown in figure A-1.

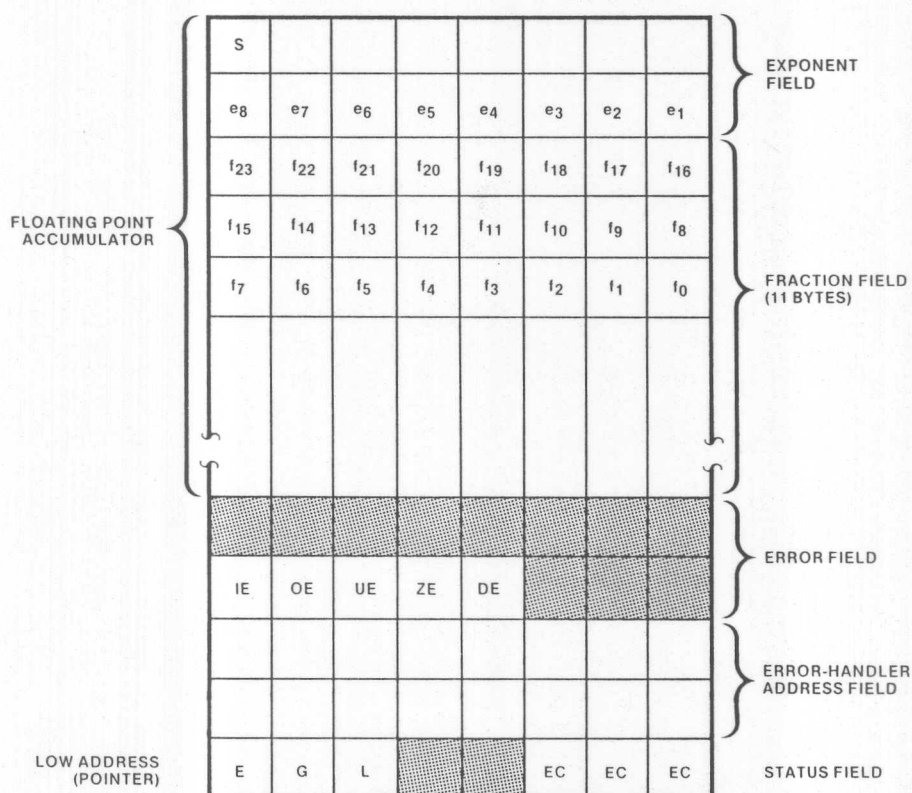


Figure A-1. Floating-Point Record Format

Status Field

Six bits are currently defined in the Status field. The setting of these bits depends on the floating-point function performed. The undefined bits are reserved for FPAL use.

The *E*, *G*, and *L* bits act as flags following a comparison (FCMPR, FZTST). A number in the FAC is compared to a second number and

- | | |
|--------------|------------------------------|
| <i>E</i> = 1 | if the FAC = second operand, |
| <i>G</i> = 1 | if the FAC > second operand, |
| <i>L</i> = 1 | if the FAC < second operand. |

The three *EC* (error condition) bits indicate whether an error just occurred. The type of error can be determined from these bit settings as follows:

Error Code	Interpretation
000	No error
001	Attempted division by zero
010	Domain error (e.g., $\sqrt{-1}$)
011	Overflow
100	Underflow
101	Invalid number in FAC
110	Invalid number in memory
111	Currently undefined

Error-Handler Address Field

The Error-Handler Address field contains the address of the error-handler subroutine. This may be the FPAL's default error handler, FERHND (described in Chapter 4), or a routine of your own. In either case, the address is loaded into this field by either the initialization subroutine (FSET) or the reset subroutine (FRESET).

Error Field

The bits in the Error field are used to accumulate error statistics. Only five bits of this field are used currently.

If any of the *IE*, *OE*, *UE*, *ZE* or *DE* bits is set, the error described below has occurred at least once since the last time the respective bit was set to zero (by the FSET or FRESET subroutine).

Bit	Interpretation
IE	Invalid operand
OE	Overflow error
UE	Underflow error
ZE	Attempted division by zero
DE	Domain error

The remaining three bits of the low-address byte are currently unused. Setting any of these bits to one causes undefined results.

Floating-Point Accumulator

The Fraction and Exponent fields shown in figure A-1 actually contain an unpacked version of the format assumed for 32-bit floating-point numbers in memory (figure A-2). The f_{23} (normalization) bit shown in figure A-1 is implied in the packed format; $f_{23}=0$ if the Exponent field is zero and otherwise $f_{23}=1$. In both figures, 's' is the 'sign' bit.

HIGH ADDRESS	S	e ₈	e ₇	e ₆	e ₅	e ₄	e ₃	e ₂
	e ₁	f ₂₂	f ₂₁	f ₂₀	f ₁₉	f ₁₈	f ₁₇	f ₁₆
	f ₁₅	f ₁₄	f ₁₃	f ₁₂	f ₁₁	f ₁₀	f ₉	f ₈
LOW ADDRESS (POINTER)	f ₇	f ₆	f ₅	f ₄	f ₃	f ₂	f ₁	f ₀

Figure A-2. Floating-Point Number Format in Memory

Two FPAL subroutines operate on 32-bit integers. FIXSD converts a floating-point number in the FAC to an integer in memory. FLTDS converts an integer in memory into a floating-point number in the FAC. The format of the 32-bit two's complement integer stored in memory is shown in figure A-3. In this figure, i₃₂ (the high-order bit) is the sign bit.

HIGH ADDRESS	i ₃₂	
LOW ADDRESS (POINTER)		i ₁

Figure A-3. Integer Format in Memory



This appendix defines terms used elsewhere in the manual along with the formulas used for rounding values and decoding exponent wraparound.

Floating-Point Zero

The word with all bits equal to zero is defined as the unique floating-point zero. No other form for floating-point zero is provided by FPAL.

Invalid Numbers

All bit patterns are valid except those described here.

The first set of invalids are those whose exponent field is set to all ones. This set is used for infinities, indefinites, pointers, etc. Infinities are defined as:

+INF 's' bit = 0; all other bits = 1
-INF all bits = 1

The indefinite form is:

IND 's' = 0; exponent bits all = 1; fraction bits = 0

A second set of bit patterns is currently defined as invalid. These are numbers whose exponent field is zero with at least one other bit set to one.

Single-Precision Format

Single-precision formats in the Floating-Point Accumulator and 8080 or 8085 memory are as shown in figures A-1 and A-2. The three fields within these formats are:

- s Sign bit. Sign-magnitude representation where s=0 means positive and s=1 means negative.
- e Exponent bits. Except in the case of floating-point zero, the exponent is offset by a *bias* of $2^7 - 1$; i.e., the stored exponent is $(2^7 - 1)$ larger than the true exponent. All zeros and all ones in the exponent field are currently reserved for the floating-point zero and the invalid numbers described above.
- f Fraction bits. When the exponent is nonzero, an implicit one bit is assumed at the left of the fraction; the binary point is between the assumed bit and the explicit fraction bit.

The number base for the FPAL is binary. The value of a given binary representation (where 's' is the sign bit, 'e' is a binary exponent value, and 'f' is a binary fraction value) can be formulated as:

$$(-1)^s \cdot 2^{e-(2^7-1)} \cdot (1. + .f) \quad \text{where } e \neq 0 \text{ and } e \neq FF$$

Rounding

If rounding is required to produce the final result of a *floating-point operation* (which does not include FQFD2B and FQFB2D), the 'round to even' rule is used. This rule returns the floating-point number closest to the true result. In the ambiguous case, i.e., when two floating-point numbers are equally close to the true result, the number with a zero in the least significant bit of the fraction is returned (that is, the nearest 'even' number is returned).

Exponent Wraparound

When overflow or underflow occurs during FPAL operations, the correct fraction results but the exponent is 'wrapped around.' This is consistent with the FPAL development philosophy that no information should be lost and that you, the user, should be able to decide what you want to do when an overflow/underflow exception occurs.

A 'wrapped around' exponent is defined to be e_w where the true (offset) exponent e_t can be derived from e_w by considering an expanded range of exponents and

on overflow	$e_t = e_w + (3 \cdot 2^8 - 2)$
on underflow	$e_t = e_w - (3 \cdot 2^8 - 2)$



APPENDIX C

SUMMARY OF FPAL PROCEDURES

Basic Operation

Table C-1 summarizes the input prerequisites of each FPAL procedure and the output returned. FERHND is not listed since it is called by other procedures, not by the user. Remember that FSET must be called before any other procedure.

Table C-1. FPAL Procedure Operation

FPAL Procedure	B,C Addresses	D,E Addresses	Result Stored at	Operation
FABS	FPR	---	FAC	$FAC \leftarrow FAC$
FADD	FPR	MEM	FAC	$FAC \leftarrow FAC + MEM$
FCLR	FPR	---	FAC	$FAC \leftarrow 0$
FCMPR	FPR	MEM	REG A	$FAC \quad MEM$
FDIV	FPR	MEM	FAC	$FAC \leftarrow FAC / MEM$
FERROR	FPR	---	REGS H,L	$REGS \ H,L \leftarrow ERROR$
FIXSD	FPR	MEM	MEM	$MEM_{int} \leftarrow FAC_{fp}$
FLOAD	FPR	MEM	FAC	$FAC \leftarrow MEM$
FLTDS	FPR	MEM	FAC	$FAC_{fp} \leftarrow MEM_{int}$
FMUL	FPR	MEM	FAC	$FAC \leftarrow FAC \cdot MEM$
FNEG	FPR	---	FAC	$0 \leftarrow 0$ otherwise, change sign of FAC
FQFB2D	MEM	Control Block	MEM	$MEM_{dec} \leftarrow FAC_{bin}$
FQFD2B	FPR	Control Block	FAC	$FAC_{bin} \leftarrow MEM_{dec}$
FRESET	B(0)=Error Handler Bit C=Error Field Initialization	User Error Handler	FPR	$ERROR \leftarrow B,C$ $ERR \ HAND \ ADDR \leftarrow D,E$
FSET	B(0)=Error Handler Bit C=Error Field Initialization	User Error Handler	FPR	$FAC \leftarrow 0$ $ERROR \leftarrow B,C$ $ERR \ HAND \ ADDR \leftarrow D,E$ $STATUS \leftarrow 0$
FSQRT	FPR	---	FAC	$FAC \leftarrow \sqrt{FAC}$
FSTAT	FPR	---	REG A	$REG \ A \leftarrow STATUS$
FSTOR	FPR	MEM	MEM	$MEM \leftarrow FAC$
FSUB	FPR	MEM	FAC	$FAC \leftarrow FAC - MEM$
FZTST	FPR	---	REG A	$FAC \quad 0$

Error Handling

Table C-2 lists the error codes set by the FPAL procedures. As was described in Chapter 4, when an error occurs a code is placed in the B-C register pair indicating which procedure was running when the error was detected, error codes are set in the

Status and Error fields of the FPR, and the error handler is called. The default error handler may perform additional operations depending on which procedure was executing.

In the case of an invalid number in the FAC, the Status field error bits and the IE bit are 'preset' by FLOAD, rather than being set by an arithmetic procedure. The call to FERHND comes from the arithmetic procedure, however.

Table C-2. FPAL Error-Handling Summary

FPAL Procedure	B,C	Status	Error Bit	Error Type	FERHND Action
FABS	9	UUUUU101	IE	FAC invalid.	No operation; FERHND returns.
FADD	1	UUUUU011	OE	Overflow.	Set FAC to largest/smallest no. (overflow positive/negative); Status=UUUUU000.
		UUUUU100	UE	Underflow.	FAC set to 0. Status set to UUUUU000.
		UUUUU101	IE	FAC invalid.	No operation; FERHND returns.
		UUUUU110	IE	Invalid no. in memory.	No operation; FERHND returns.
FCLR	—	---	---	No error conditions.	---
FCMPR	6	000UU101	IE	FAC invalid	If operands identical, Status set to 100UU101; otherwise Status is 000UU101.
		000UU110	IE	Invalid no. in memory	
FDIV	4	UUUUU001	ZE	Attempted division by 0.	FAC set to indefinite (s=0, e=1, f=0); Status set to UUUUU101; IE set.
		Others same as FADD.	Others same as FADD.	Same as FADD.	Same as FADD.
FERROR	—	---	---	No error conditions.	---
FIXSD	5	UUUUU011	OE	FAC no. too large.	Set memory to largest/smallest integer from FAC (overflow positive/negative); Status= UUUUU000.
		UUUUU101	IE	FAC invalid; integer stored is undefined.	No operation; FERHND returns.
FLOAD	—	UUUUU101	IE	Number loaded into FAC is invalid.	Not called.
FLTDS	—	---	---	No error conditions.	---
FMUL	3	Same as FADD.	Same as FADD.	Same as FADD.	Same as FADD.
FNEG	8	UUUUU101	IE	FAC invalid.	No operation; FERHND returns.
FQFB2D	—	UUUUU101	IE	FAC invalid.	Not called. Decimal record sign and D ₂ ...D _n set to *; D ₁ set to: '+' if FAC = + INF '-' if FAC = -INF '?' if FAC = IND '0' if FAC = -0 '*' all other invalids.
FQFD2B	—	---	---	No error conditions.	---

Table C-2. FPAL Error-Handling Summary (Cont'd.)

FPAL Procedure	B,C	Status	Error Bit	Error Type	FERHND Action
FRESET	—	---	---	None, but if MA, UO, or OO bits = 1, results are undefined.	---
FSET	—	---	---	Same as FRESET.	---
FSQRT	10	UUUUU101 UUUUU010	IE DE	FAC invalid. Domain error (FAC negative).	No operation; FERHND returns. FAC set to indefinite (s=0, e=1, f=0); Status set to UUUUU101; IE set.
FSTAT	—	---	---	No error conditions.	---
FSTOR	—	---	---	No error conditions.	---
FSUB	2	Same as FADD.	Same as FADD.	Same as FADD.	Same as FADD.
FZTST	7	UUUUU101	IE	FAC invalid.	No operation; FERHND returns.

Procedure Sizes

Table C-3 summarizes size information for each FPAL procedure (in bytes). These absolute figures must be read against the context of FPAL operation as a whole, however, as detailed in the notes following this table.

Table C-3. FPAL Procedure Sizes

FPAL Procedure	Bytes	Subroutines Linked
FABS	36	None
FADD/FSUB	463	FCLR, FLOAD, FNEG, Support Routines
FCLR	21	None
FCMPR	159	Support Routines
FDIV	342	Support Routines
FERHND	237	FCLR, FLOAD
FERROR	10	None
FIXSD	178	None
FLOAD	88	None
FLTDS	139	FCLR, Support Routines
FMUL	404	FCLR, Support Routines
FNEG	43	None
FQFB2D	1585	FADD, FABS, FMUL, FSTAT, FDIV, FIXSD, FCMPR, FZTST
FQFD2B	725	FMUL, FNEG, FDIV, FLTDS
FRESET	42	FERHND
FSET	57	FERHND
FSQRT	247	None
FSTAT	1	None
FSTOR	35	None
FZTST	56	None
Support Routines	259	None

NOTES

1. FSET must be used. Since it links in FERHND and FERHND links in FCLR and FLOAD, the total space requirement for FSET is

FSET	57
FERHND	227
FCLR	21
FLOAD	88
	<hr/>
	393 bytes

Since FRESET links in the same subroutines as FSET, they need not be counted again if FRESET is specified.

FRESET	40 bytes
--------	----------

2. A number of arithmetic procedures (FADD, FSUB, FDIV, FMUL, FCMR, and FLTDS) link in a set of *FPAL support routines*. These routines need be linked and counted only once.

Support Routines	259 bytes
------------------	-----------

3. Calling FADD or FSUB causes both subroutines to be linked into your program. These subroutines link in FCLR, FLOAD, and the support routines – all of which have been previously counted. In addition, FNEG is linked, so that the additional space requirement for FADD/FSUB becomes

FADD/FSUB	463
FNEG	43
	<hr/>
	506 bytes

4. FDIV and FCMR link in only the FPAL support routines. FMUL and FLTDS link in only the support routines and FCLR, both of which are already counted. Thus, only the absolute count for these procedures need be considered.
5. FABS, FERROR, FIXSD, FSTAT, FSTOR, and FZTST link in no other procedures and only their absolute sizes need be considered.
6. FCLR, FERHND, FLOAD, and FNEG are all linked by other subroutines and included in those subroutines' total byte count. They need not be counted again if referenced separately.

Example:

To compute $I = \text{FIXSD}(A * B)$, you must allow space for:

FSET	393
FERHND	---
FLOAD	---
FMUL	404
FCLR	---
Support Routines	259
FIXSD	178
	<hr/>
	1234 bytes

Procedure Timing

When computing execution speeds of FPAL procedures, you must be even more wary of absolutes than when computing size requirements. We could list the following times for the basic arithmetic operations:

FADD	0.7 milliseconds
FSUB	0.7
FMUL	1.5
FDIV	3.6
FCMR	0.3

These figures are only approximations, however, and the actual figure for a given operation depends on the operands involved. The following examples illustrate this point.

Example 1

Operand 1: 40000000H

Operand 2: 40000000H

Procedure	ms
FADD	0.69
FSUB	0.79
FMUL	1.48
FDIV	3.79
FCMPR	0.33

Example 2

Operand 1: 41C80000H

Operand 2: 41F00000H

Procedure	ms
FADD	0.70
FSUB	0.83
FMUL	1.43
FDIV	3.60
FCMPR	0.28

Example 3

Operand 1: 41C8FF00H

Operand 2: 41F0F0FFH

Procedure	ms
FADD	0.66
FSUB	0.83
FMUL	1.54
FDIV	3.60
FCMPR	0.28

Example 4

Operand 1: 3FFFFFFFH

Operand 2: 3FFFFFFEH

Procedure	ms
FADD	0.65
FSUB	1.62
FMUL	1.66
FDIV	3.61
FCMPR	0.32

NOTE

The only reason FSUB appears to take longer than FADD in these examples is that all operands are positive. On the average, both will take the same time since they are simply different entry points into the same subroutine.